APPENDIX I

InQuira Match Language ("IML")

The InQuira Match Language (IML) is a language for specifying conditions and actions based on matching words, phrases, and concepts in user requests and application content. It is intended be used as a means of matching natural language sentences, whether user questions or sentences within indexed content. During exemplary request processing, a rules engine can process IML within rule components, which specify conditions. Then, the rules engine can process IML expressions that specify actions for rules containing true conditions for the request being processed. IML is a regular expression language, comprising a set of symbols that define various functions and operations. These symbols are specified in combination with words of interest to form IML expressions.

Arguments for expressions are delimited by parentheses, and separated by commas. A programmer can create nested expressions using parentheses to delimit the enclosed expressions.

The IML Elements are:

- Basic Expressions
- Operators
- Keywords
- Macros
- Number Expressions
- Variables

I. Basic Expressions

Basic IML expressions are the means of specifying semantic matching for words and phrases in user requests and in indexed content, such as that stored in a Semantic Index. Basic expressions generally resolve to single units of meaning within user requests or indexed content. Within indexed content, single units of meaning occupy a designated position, and are referred to as "offsets." The various types of base expressions enable one to match words and phrases as literal character strings, canonical forms that include variations in wordform and punctuation, and concepts that include semantic relations as defined in an ontology.

The following basic expressions are valid in IML:

- Literal Expressions
- Canonical Form Expressions
- Concept Expressions.

A. Literal Expressions

A literal expression matches only the exact specified character string. Literal expressions are sensitive to case, punctuation, and spacing, and are specified by enclosing the word or phrase within double quotes, in the form: "some string."

B. Canonical Form Expressions

Canonical form expressions match the specified string and any variations in inflection or form, such as capitalization, tense, or other valid morphological variations. Canonical form expressions are specified by either (1) a non-quoted string beginning with an upper- or lower-case letter, succeeded by any alphanumeric characters, with no punctuation or spaces, or (2) any alphanumeric string enclosed within single quotes, including punctuation and spaces. The expression 'cat' matches cat, Cat, cats, Cats, and cat's, but not catalog.

C. Concept Expressions

Concept expressions match occurrences of the specified concept and its synonyms and are specified by the concept name within angle brackets. Concept names consist of three identifiers, separated by periods, such as: <pos.domain.headword>. Here, "pos" specifies the concept's part of speech; "domain" specifies the domain to which the concept is assigned; and "headword" specifies the concept headword, which is an indicator for the collection of specified synonyms and other relationships that define the concept. As an example, the expression <noun.animal.cat> matches Cat, cats, and feline, but not catalog.

II. Operators

IML operators specify the relevant portion, or "range," of a request (e.g., during query processing) or a content source, such as a document, (during semantic indexing) over which to apply an IML expression. To perform an action, the rules engine matches expressions within the specified portion of the document or request. For example, you can specify operators to apply expressions to:

- sentences within documents
- entire documents
- sentences within documents having titles that match a specified expression
- combining operators

Range Operators specify the scope, or amount of surrounding text that will be associated with specified expressions. Examples of ranges include phrase, sentence, and proximity within a specified number of words. For example, a literal expression "cat" will match any occurrences of the string cat. A sentence range operator specifies that this expression will apply to, or match, sentences that contain the string "cat". A document range operator specifies that this expression will match documents that contain the string cat.

The phrase and proximity Range Operators apply to both conditions (e.g., question pattern) and actions (search component and direct IML expressions). The sentence, title, subtitle, section, document, and reference Range Operators typically only apply to actions (search component and direct IML).

A. Phase Range Operator

The phrase (PHRASE) range operator returns phrases that contain all of the expressions specified as arguments. For example, PHRASE(<cat>,<hat>) matches phrases containing both of the concepts (or synonyms of) "cat" or "hat."

B. Sentence Range Operator

The sentence (SENT) range operator returns sentences that contain all of the expressions specified as arguments. SENT is valid only within Rule actions.

Sentence Range Operator	
Syntax: SENT([expression]{separator expression})	A programmer specifies the sentence range operator in this form
where:	
expression	specifies a valid IML expression as an argument that the range operator will apply to
separator	specifies an argument separator, if required
combining operator	specifies a combining operator, if required
Example:	
Specify SENT(<cat>,<hat>)	**To match** sentences containing both of the concepts (or synonyms of) cat or hat

C. Title Range Operator

The title (TITLE) range operator returns documents having titles that contain all of the expressions specified as arguments. Document titles are determined during semantic indexing. TITLE is valid only within Rule actions.

Title Range Operator	
Syntax: TITLE([expression]{separator | combining operator}{expression})	A programmer specifies the title range operator in this form
where:	
expression	specifies a valid IML expression as an argument that the range operator will apply to
separator	specifies an argument separator, if required
combining operator	specifies a combining operator, if required
Example:	
Specify TITLE(<cat>,<hat>)	**To match** documents with titles that contain both of the concepts (synonyms of) cat or hat

D. Subtitle Range Operator

The subtitle (SUBTITLE) range operator returns document sections having applied subtitles that contain all of the expressions specified as arguments. Document subtitles are determined during semantic indexing. SUBTITLE is valid only within Rule actions.

Subtitle Range Operator	
Syntax: SUBTITLE([expression]{separator | combining operator} {expression})	The programmer specifies the subtitle range operator in this form
where:	
expression	specifies a valid IML expression as an argument that the range operator will apply to
separator	specifies an argument separator
combining operator	specifies a combining operator, if required
Example:	
Specify SUBTITLE(<cat>,<hat>)	***To match*** document sections having subtitles that contain both of the concepts (synonyms of) cat or hat

E. Section Range Operator

The section (SECTION) range operator returns document sections that contain all of the expressions specified as arguments. Document sections are determined during semantic indexing. SECTION is valid only within Rule actions.

Section Range Operator	
SECTION([expression]{separator | combining operator{expression})	The programmer specifies the section range operator in this form
where:	
Expression	specifies a valid IML expression as an argument that the range operator will apply to
Separator	specifies an argument separator
combining operator	specifies a combining operator, if required
Example:	
Specify SECTION(<cat>,<hat>)	***To match*** document sections that contain both of the concepts (synonyms of) cat or hat

F. Document Range Operator

The document (DOC) range operator returns documents that contain all of the expressions specified as arguments. Documents are defined during semantic indexing. A user's question is also considered a document for purposes of this operator.

Document Range Operator	
Syntax: DOC([expression]{separator expression})	The programmer specifies the title range operator in this form

where:	
expression	specifies a valid IML expression as an argument that the range operator will apply to
separator	specifies an argument separator, if required
combining operator	specifies a combining operator, if required
Example:	
Specify DOC(<cat>,<hat>)	***To match*** documents containing both of the concepts (synonyms of) cat or hat

G. Proximity Range Operator

The proximity (NEAR) range **operator** returns a specified range of words that contains all of the **expressions** specified as arguments. The programmer specifies the range as a parameter, n. NEAR is valid within Rule conditions and actions.

Proximity Range Operator	
Syntax: NEAR_n([**expression**]{separator expression})	The programmer specifies the proximity range operator in this form
where: **n** specifies the number of offsets that defines the range. Offsets indicate unique index positions, approximately equal to single words	
expression	specifies a valid IML expression as an argument that the range operator will apply to
separator	specifies an argument separator, if required
combining operator	specifies a combining operator, if required
Example:	
Specify NEAR_5(<cat>,<hat>)	***To match*** content that includes both of the concepts (synonyms for) cat or hat within a 5 word range

H. Reference Range Operator

The reference (REFERENCE) range **operator** returns documents having hypertext links to them that contain all of the expressions specified as arguments. REFERENCE is valid only within Rule actions.

Reference Range Operator	
Syntax: REFERENCE([**expression**]{separator | combining operator}{expression})	The programmer specifies the reference range operator in this form
where:	
expression	specifies a valid IML **expression** as an argument that the range operator will apply to

separator	specifies an argument separator, if required
combining operator	specifies a combining operator, if required
Example:	
Specify REFERENCE(<cat>,<hat>)	***To match*** documents having hypertext links pointing to them that contain both of the concepts (synonyms of) cat or hat

I. Combining Operators

Combining Operators specify operations on expressions or their associated ranges. Combining Operators are used to match an area of content that is defined as the result of an operation on two or more specified ranges. For example, an intersection combining operator can be configured to return only these sentences that include both of two specified concepts. Combining Operators are specified inline, between the expressions on which they operate. The expressions that are combined can be any valid IML expressions. Combining Operators are either binary or have Boolean (and/or) semantics.

J. Intersection Operator

The intersection (IS) combining operator specifies the intersection of the ranges of the specified expressions.

Intersection Operator	
Syntax: {expression} IS {expression}	The programmer specifies the intersection combining operator in this form
where:	
expression	specifies a valid IML expression that the operator will apply to
Example:	
Specify SENT(<cat>) IS PHRASE(<hat>)	***To match*** sentences that contain the concept <cat>, which are also sentences which contain the concept hat.

K. Union Operator

The union (OR) combining operator specifies the union of the ranges of the specified expressions.

Union Operator	
Syntax: {expression} OR {expression}	The programmer specifies the union combining operator in this form
where:	
expression	specifies a valid IML expression that the

	operator will apply to
Example:	
Specify SENT(<cat>OR<hat>)	***To match*** sentences that contain either of the concepts (synonyms of) cat or hat

L. Difference Operator

The difference (ISNT) combining operator specifies the difference of ranges of the specified expressions.

Difference Operator	
Syntax: {expression} ISNT {expression}	The programmer specifies the difference combining operator in this form
where:	
expression	specifies a valid IML expression that the **operator** will apply to
Example:	
Specify SENT(<cat>) ISNT SENT(<hat>)	***To match*** sentences that contain the concept (synonyms of) cat, excluding sentences that also include the concept hat

M. Offset Difference Operator

The difference offset (WITHOUT) combining operator specifies the range of difference of the **specified expressions**.

Offset Difference Operator	
Syntax: {expression}WITHOUT {expression}	The programmer specifies the difference offset operator in this form
where:	
expression	specifies a valid IML expression that the operator will apply to
Example:	
Specify DOC(<cat>) WITHOUT SENT(<hat>)	***To return*** offset ranges from documents that contain the concept (synonyms of) cat, excluding sentence ranges that include the concept hat

N. Offset Intersection Operator

The overlap (OVERLAP) combining operator specifies the ranges representing the offset overlap of all arguments. The overlap operator returns the portion of the text that the specified expressions have in common.

Offset Intersection Operator	
Syntax: {expression}OVERLAP{expression}	The programmer specifies the overlap range operator in the form
where:	
expression	specifies a valid IML expression that the operator will apply to
Example:	
Specify SUBTITLE(cat) OVERLAP hat	***To match*** occurrences of hat located within section offset ranges that have cat in their subtitles

A. III. Keywords

Keywords perform specific matching functions and are used to limit matching for an expression to the specified characteristic. For example, one can specify keywords to represent any single word, or to match an expression only if the matching offset is the first word in a document. One can specify keywords inline within the expression to which they apply. The following keywords are valid in IML:

- WORD, which matches any, but exactly one, token
- BEGIN, which matches the first word in a document
- END, which matches the last word in a document
- THIS, which assigns a concept sense to one or more tokens in IML expressions within the Dictionary

A. WORD Keyword

The WORD keyword matches any, and exactly one word.

WORD Keyword	
Syntax: [expression] WORD [expression]	The programmer specifies the WORD keyword in this form
where:	
expression	specifies any valid IML expression
Example:	
Specify this WORD house	***To match*** any single word within a matched **expression**, for example*:* this old house this red house **And not match** this big old house, this house Note: You can modify the WORD keyword to match multiple words using number

	expressions, as described in Section IV.

B. BEGIN Keyword

The BEGIN keyword specifies the beginning of a document, prior to the first word. Use the BEGIN keyword to limit matching for an expression to the first word in a document.

BEGIN Keyword	
Syntax: BEGIN expression	The BEGIN keyword is specified in this form
***where:* BEGIN** specifies to match prior to the first word in a document, and **expression** specifies any valid IML expression	
expression	specifies any valid IML expression
Example:	
Specify DOC (BEGIN <cat>)	**To match** documents having the concept (synonyms of) cat as the first word in the body of the document

C. END Keyword

The END keyword specifies the end of a document, after the last word. Use the END keyword to limit matching for an expression to the last word in a document.

END Keyword	
Syntax: END expression	The programmer specifies the END keyword in this form
***where:* END** specifies to match only the first word in a document	
expression	specifies any valid IML expression
Example:	
Specify DOC (<cat> END)	***To match*** documents having the concept (synonyms of) cat as the last word in the body of the document

D. THIS Keyword

The THIS keyword is valid only in IML expressions within concept definitions in the Dictionary. The THIS keyword specifies a part of an IML expression that represents the defined concept, and limits the part of expression that the rules engine will use as the concept in subsequent operations.

THIS Keyword	
Syntax:	The programmer specifies the THIS keyword

expression=THIS	in this form
where: **THIS** specifies the portion of the expression to use as the concept in subsequent operations	
expression	specifies any valid IML expression that defines the concept
Example: Defining the concept <online_banking>	
Specify NEAR_5((online OR virtual), banking=THIS)	***To define*** a concept <online banking> ***That matches*** portions of text like: "banking online is fun and easy..." "try banking the virtual way with our..." ***And specifies that*** the rules engine will use the word banking in subsequent operations on this concept

B. IV. Number Expressions

Number expressions specify a number or range of occurrences to match the expression to which they apply. Number expressions are applied to numeric ranges for any valid IML expressions. For example, one can specify to match one or more occurrences of an expression, or up to five occurrences (for example) of a specified expression. The following number expressions are valid in IML:

- simple range number expressions.
- ascending range number expressions.
- descending range number expressions.

General Rules for Specifying Number Expressions

A user specifies number expressions as a single range of integers separated by a hyphen and enclosed within parentheses. Number expressions follow the expression to which they apply.

Numerical Expression	
Syntax: expression(number_expression)	A user specifies number expressions in this form
where:	
Number_expression	specifies a valid number expression
expression	specifies any valid IML expression

A. Simple Range

The simple range number expression specifies a range that spans the specified lower and upper boundaries.

Simple Range	
Syntax: expression(n-m)	A programmer specifies the simple range number expression in this form
where: **n** specifies the lower limit of the range **m** specifies the upper limit of the range, where the value of "m" must be greater than or equal to the value of "n"	
expression	specifies any valid IML expression
Example:	
Specify <hat>(2-3)	***To match*** occurrences of the concept (synonyms of) hat in a series of two or three: hat hat hat hat hat ***And not match*** single occurrences of the term hat **Note:** In the match example, the rules engine would produce a total of four matches: • a match for the series of two hats on the first line: {hat hat) • a match for the first series of two hats on the second line: {hat hat) hat • a match for the second series of two hats on the second line: hat {hat hat} • a match for the series of three hats on the first line: (hat hat hat)

B. Ascending Range

The ascending range number expression specifies all integers greater than or equal to the specified lower boundary.

Ascending Range	
Syntax: expression (n-) expression	A programmer specifies the ascending range number expression in this form
where: **n** specifies the lower limit of the range expression	
expression	specifies any valid IML expression
Example:	
Specify cat WORD(2-) hat	**To match** occurrences of the specified expressions cat and hat having intervening words within the specified range or two or more: cat and the hat cat and the red hat ***And not match*** occurrences of the specified

	expressions cat and hat having intervening words beyond the specified range of two or more: cat and hat

C. Descending Range

The descending range number expression specifies all integers less than or equal to the specified upper boundary, including 0.

Descending Range	
Syntax: **expression**(-n)	A programmer specifies the descending range number expression in this form:
where: **n** specifies the lower limit of the range	
expression	specifies any valid IML expression
Example:	
Specify cat WORD(-2) hat	***To match*** occurrences of the specified expressions cat and hat having intervening words within the specified range or two or fewer: cat hat cat and hat cat and the hat ***And not match*** occurrences of the specified expressions cat and hat having intervening words beyond the specified range of two or fewer: cat and the red hat

V. Macros

Macros are assigned character substitutions for commonly specified IML **expressions**. Macros may be defined by the user as an arbitrary valid IML expression. In addition, several macros are included as part of the IML language.

*: specifies 0 or more occurrences of keyword WORD
+: specifies 1 or more occurrences of keyword WORD

A programmer defines new macros inline within IML expressions, in the form:
{expression}macro{expression}

A. Zero or More Words Macro

Zero or More Words Macro

Syntax: *	A programmer specifies the macro * to match occurrences of zero or more words, in this form
where: symbol "*" matches occurrences of one or more words	

Note: The * macro can be expressed in IML as the following keyword and number expression: WORD(0-)

B. One or More Words Macro

One or More Words Macro	
Syntax: +	A programmer specifies the macro "+" to match occurrences of one or more words, in this form
where: symbol "+" matches occurrences of one or more words	

Note: The + macro can be expressed in IML as the following keyword and number expression: WORD(1-)

C. VI. Variables

A variable is a symbol that represents a contiguous set of words in a user request. Variables are a method of associating a part of the user request with an expanded (more general) or reduced (more specific) set of meanings. There are local and global variables. Local variables apply only within the rule in which they are specified. Global variables apply within the rule in which they are set, and also within subsequent rules.

The rules engine processes variables during request processing. When a rules engine evaluates a rule as true, it sets any variables specified within that rule. Once a variable is set, its value is substituted when it is referenced from another expression. Local variables can be referenced only by expressions within the same rule (but not from within the expression in which they are set). Global variables can be referenced by any subsequent rules.

Note: a global variable cannot be referenced in a rule that precedes the one in which its value is set. The rules engine checks global variables for validity, and will issue a warning if this occurs. Variables are specified by:

- specifying them within IML expressions
- assigning their values
- referring to them

A. Specifying Variables

Specifying Variables	
Syntax: (expression)=variable or (expression)=#variable	A programmer specifies a variable within an IML expression in this format
where: symbol " #" specifies a global variable, which applies to all subsequent rules. Omitting the # prefix specifies a local variable, which applies only to the current rule	
variable can be any alphanumeric string	The first character in the string must be a letter (alpha) character. Letter characters can be upper and lower case. If only upper case characters are specified, the programmer can also use the - (hyphen) character. Global variables can also include the characters: • _ (underscore) • - (hyphen) • $ • %
Example: expression=A	
Specify	You can specify the same string for multiple variables. If the programmer specifies the same string for multiple global variables, the Rules Engine adds each assignment as it is processed.

B. Retrieving Values from Variables

A programmer retrieves the value from a variable using a Variable Instantiation Language (VIL) function. When retrieving the value of a variable, the programmer is to use various VIL functions to determine its value, which *instantiate* it in the context of an individual user request. The programmer specifies these functions using VIL expressions, usually within a rule action. VIL expressions contain one or more function calls, and each function call has one or more optional parameters. (A full specification of VIL is not included here.)

C. Referring to Variables

A programmer can include VIL expressions to refer to variables from within index queries, custom content, SQL queries, or any other expressions or type of expression. The programmer specifies references to variables within curly brackets {}, in the form:

{VIL_expression}
where: {} delimits VIL expressions contained within text, such as IML expressions or custom content. For example, a programmer can use VIL expressions to refer to variables from within custom content specified in the answer section of a rule:

You can buy a {VIL_expression_1} today for {VIL_expression_2}.
where:
VIL_expression_1 refers to a variable set in a rule condition that resolves to a product name or type of product mentioned in a user request that matched a price of product rule. **VIL_expression_2** refers to a variable instantiated from a database query for the price of the product mentioned in the user request. The instantiated statement displayed by the User Interface might be: You can buy a *Model 500 washer/dryer* today for *$900.*

A programmer can also specify a variable to resolve to a valid IML expression, which the rules engine then evaluates as an action.